Progressive Isosurface Extraction from Tetrahedral Meshes

U. Labsik P. Kipfer S. Meinlschmidt G. Greiner

Computer Graphics Group, University of Erlangen-Nuremberg,
Am Weichselgarten 9, 91058 Tennenlohe, Germany
Tel: +49-9131-8529919, Fax: +49-9131-8529931
Email: labsik@cs.fau.de

Abstract

In this paper we present a technique for transforming a tetrahedral mesh into a progressive representation based on half edge collapses. This allows the efficient transmission of the mesh from a remote computer where the simulation was computed to a visualization computer. During the transmission the user can start visualizing while the transmission is still in progress. We show a technique for progressively extracting isosurfaces from the progressive mesh. Starting with the base mesh an isosurface for a specific value is computed and will locally be improved where a vertex is inserted to the mesh

Keywords: *tetrahedra, isosurfaces, progressive meshes, grid reduction*

1. Introduction

One of the most used methods to visualize 3D scalar fields is the extraction of isosurfaces. An isosurface is a region in the field with a constant scalar value, defined as $S(\lambda) = \{\mathbf{x} \in \mathbb{R}^3 | F(\mathbf{x}) = \lambda\}$. In this paper we focus on scalar fields decomposed into tetrahedra which form a tetrahedral grid. In a standard approach all cells which are intersected by the isosurface have to be determined. This process is called *cell search*. For these cells the isosurface has to be computed, i. e., a *cell triangulation* has to be found.

For simulation purposes the tetrahedral grids can become very large. This is essential due to the required numerical accuracy of the simulation. Therefore it is often necessary to use parallel remote computers to perform the simulation while using a graphics workstation as visualization front-end. For the visualization a variety of strategies are possible. A flexible approach is to transfer the tetrahedral mesh to the workstation in a suitable way and to use this grid for several visualization methods like the isosurface extraction. A main drawback of this approach is that the mesh has to be transferred completely to the workstation before the user can start to visualize the scalar field.

To overcome this problem, in this paper we present an approach to transform the tetrahedral grid into a progressive representation which can be used to extract isosurfaces or use other visualization techniques like slices or volume rendering on continuous levels of detail. We exploit grid simplification techniques based on the half edge collapse operator and transform the tetrahedral mesh into a progressive tetrahedral mesh [17].

To achieve acceptable visualization results already on the coarse approximations of the tetrahedral grid, it is essential to keep track of the error originating from the grid reduction. Each half edge collapse adds an approximation error to the mesh. This error can be divided into different components. We distinguish between two different error types, the geometric and the field error. By using a technique based on error quadrics [6], the reduction algorithm is steered to keep both error components minimal.

Based on the progressive tetrahedral representation we present an algorithm to extract isosurfaces progressively from the tetrahedral mesh. By using this technique, visualizing the field data can be started after the coarse base mesh is transferred to the visualization workstation. The quality of the isosurface improves continuously during the transmission of the grid. We also combine a standard approach for fast isosurface extraction [2] with our approach to enable a fast recomputation of the isosurface after the isovalue was changed by the user.

This paper is organized as follows. In Section 2 we give a brief overview of existing methods for simplifying tetrahedral meshes and extracting isosurfaces. We explain our mesh reduction approach based on half edge collapses in Section 3 and present our steering criteria for the approximation error in Section 3.3. After explaining the progressive isosurface extraction algorithm in Section 4 we discuss results of our algorithm in Section 5 and finish with a conclusion.

2. Related Work

In this paper we mainly deal with the topics of mesh reduction of tetrahedral grids, progressive transmission and the progressive extraction of isosurfaces from the grid. In this section we discuss related approaches in the field of mesh reduction on 2D and 3D meshes and methods for isosurface extraction on 3D grids.

2.1. Mesh reduction

Most of the work in the field of grid reduction was done on triangular meshes. Hoppe [10] presents an algorithm based on the edge collapse operator to simplify triangle meshes. At each step the edge with the lowest cost is collapsed and the information to restore the edge collapse by a vertex split is stored. By this algorithm a progressive mesh consisting of a coarse base mesh and a large number of vertex splits is computed. This can for example be used for progressive transmission of the mesh.

Garland and Heckbert [6] show an efficient way to keep track of the approximation error of the simplification process by using error quadrics. Each vertex is associated with an error quadric which is computed as the sum of squared differences to a set of planes. When collapsing a vertex into another one, both error quadrics are simply added. Therefore the algorithm is very fast and does not require much additional memory, only 10 coefficients per vertex. In [7] the authors present an extension of their method for meshes with surface properties like color or texture coordinates. For such meshes Hoppe [11] presents an alternative approach which leads to visually slightly better results and needs less memory than the method in [7].

Grid reduction algorithms for tetrahedral grids based on the edge collapse operator were presented in [17] and [1]. Here, the requirements for an edge collapse on a tetrahedral mesh are described, so that the mesh is still topologically correct after an edge collapse and has no elements with negative volumina and no self intersections. The papers differ in the way the approximation error is computed. While in [17] a simple method is explained to compute the error, Cignoni et al. compare different algorithms for computing the approximation error.

2.2. Isosurface extraction

We divide the related work in two different areas, grid reduction and isosurface extraction. The most popular algorithm for isosurface extraction is the *Marching Cubes* algorithm proposed by Lorensen and Cline. It works for regular hexahedral grids, traverses each cell and checks if it is intersected by the isosurface. The algorithm works with 15 base cases for the triangulation of a cell, but introduces some topological inconsistencies on certain configurations. One way to overcome these problems is to split each hexahedral cell into a number of tetrahedra and extract triangulated isosurfaces within each tetrahedron cell [3, 9].

In large meshes most of the cells are not intersected by an isosurface for a specified isovalue. Therefore techniques were developed to be able to efficiently search for cells which contribute to the surface. The *Sweeping Simplices* algorithm [16] by Shen and Johnson uses a minimum and a maximum sorted list with links from the min-sorted to the max-sorted list and dirty flags in the max-sorted list. To further improve the performance, Livnat et al. [13] presented an algorithm operating on the 2D min-max *span space*. In the span space each tetrahedron is represented as a point in a 2D domain where all tetrahedra which are intersected by the isosurface for a given value lie in a rectangular region. To speed up the search in the span space Cignoni et al. [2] presented an approach to use *interval trees*. This also improves the worst-case performance of the isosurface computation.

A different approach for mesh reduction and progressive isosurface extraction was presented by Grosso and Ertl [8]. In their algorithm, a multilevel finite element approximation of a given volume dataset is computed. A coarse base mesh is refined adaptively until the global approximation error of the mesh is lower than a given threshold. Based on this multilevel approximation they present an algorithm to progressively extract isosurfaces. Another algorithm for the adaptive reconstruction of isosurfaces from volume data was presented by Westermann and Kobbelt [18]. This method was extended in [4] to be used for web-based isosurface extraction.

3. Mesh Reduction

The isosurface extraction algorithm in this paper is based on a progressive representation of a tetrahedral mesh $\mathcal{M}$. The idea is to find a coarse base mesh $\mathcal{M}^0$ in which vertices are inserted until the original mesh $\mathcal{M}^n = \mathcal{M}$ is completely reconstructed. To compute such a coarse base mesh, a grid reduction technique is used which iteratively removes vertices from the mesh and retriangulates the hole originated by the vertex deletion. By doing so, a series of tetrahedral meshes $\mathcal{M} = \mathcal{M}^n, \mathcal{M}^{n-1}, \mathcal{M}^{n-2}, \ldots, \mathcal{M}^0$ is computed.

3.1. Half edge collapse

Several strategies exist to remove a vertex from the mesh. The most common operator for the simplification of triangle meshes is the well known *edge collapse* operator [10]. It removes one vertex and two triangles by merging two adjacent vertices and thus defines the transition from a mesh $\mathcal{M}^i$ to $\mathcal{M}^{i-1}$. A position for the merged vertex can be found by minimizing an error functional.

One of the advantages of the edge collapse in contrast to other mesh simplification techniques is the simplicity to inverse the coarsening process. This is done by splitting the vertex into two, and inserting two additional triangles. This operator is called *vertex split*. Both the edge collapse and the vertex split are shown in Figure 1.

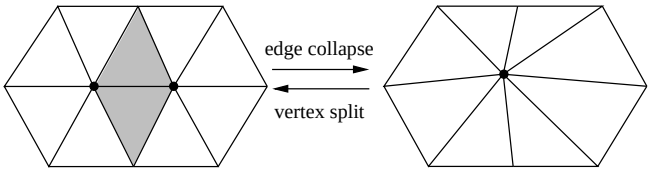

Figure 1. The edge collapse and the inverse vertex split operator.

To be able to restore the original mesh by a number of vertex split operations, some data has to be stored for every single vertex split. This is the index of the start vertex in mesh $\mathcal{M}^{i-1}$, the indices of the two vertices of collapsing triangles which are adjacent to the start vertex and the position of the start/end vertex. Altogether this are three integers and six float values for every vertex split.

A restriction and simplification of the edge collapse is the so-called *half edge collapse*. Here, one vertex is moved into the position of the other vertex, see Fig. 2. This operator has two advantages. On the one hand, the computation of the hierarchy becomes much easier because no optimization is necessary for finding the new vertex positions, on the other hand only one vertex position has to be stored in the progressive representation of the mesh. Only three integers and three floats are stored. This makes the progressive mesh representation based on half edge collapses more efficient than a shared vertex representation of the triangle mesh.

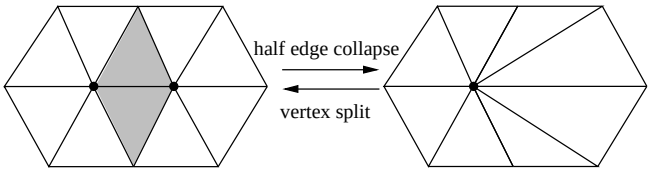

Figure 2. The half edge collapse and the inverse vertex split operator.

Up to now the operators shown work on triangle meshes. In the case of tetrahedral meshes these operators nearly look the same. For the half edge collapse one vertex $\mathbf{p}_i$ is moved into the position of an adjacent vertex $\mathbf{p}_j$ along the edge between the vertices. This removes the edge and all n tetrahedra around this edge. We name this operation $hec(\mathbf{p}_i \rightarrow \mathbf{p}_j)$. To be able to restore the position and the scalar value of the removed vertex after the half edge collapse, the index of the removed vertex and the indices of the vertices around the edge have to be stored. This are four float values and $n+1$ integer values for one vertex and n tetrahedra. This again needs less information than the normal shared vertex representation.

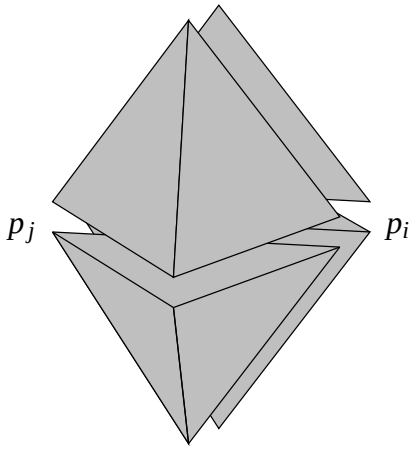

Figure 3. Configuration of 4 tetrahedra around the edge from $\mathbf{p}_j$ to $\mathbf{p}_i$. These tetrahedra will be deleted by the half edge collapse.

3.2. The reduction algorithm

Up to now we have defined the topological operators to delete a vertex from the mesh. To compute the coarse approximation $\mathcal{M}^0$ of the given mesh $\mathcal{M}$ a sequence of half edge collapses has to be specified.

The basic mesh simplification algorithm now works as follows. A weight function is defined which associates a weight for each half edge of the mesh. This has to be done, because in general the weight $\Delta E(hec(\mathbf{p}_i \rightarrow \mathbf{p}_j))$ will be different to $\Delta E(hec(\mathbf{p}_j \rightarrow \mathbf{p}_i))$. Then, all half edges of the mesh are sorted into a priority queue according to their weight. The first edge is taken from the queue. It has to be checked whether it is forbidden to collapse the edge. This can have different reasons which were already extensively discussed in [17]. It is not allowed to collapse an edge with a start vertex on the boundary of the mesh if the edge itself is not on the boundary. This check can be performed very fast because the flags whether points or edges are on the boundary can be precomputed.

The other class of tests is more time intensive. They have to be performed on the configuration of the tetrahedra after

each collapse. The first one is the test for flipped or folded tetrahedra. This can be done by checking the volumina of the tetrahedra around the remaining vertex of the collapse operation. The volume of a tetrahedron T_{abcd} is computed by

$$V_T = \frac{1}{6}\det(b-a, c-a, d-a) \quad (1)$$

If one of the volumina becomes negative the collapse operation is not allowed.

The second problem of this class are self intersections of the boundary of the tetrahedral mesh. This test is very expensive. In our implementation we try to avoid this test. This is possible for most tetrahedral meshes if the boundary geometry is taken into account when computing the edge costs for a half edge collapse.

If all tests are passed and the edge is allowed to be collapsed the half edge collapse is performed and its entry in the priority queue is deleted. By the collapse the weights for edges in the neighborhood of the deleted edge may have changed and have to be recomputed. Afterwards the priority queue has to be updated.

This algorithm will be performed as long as the priority queue is not empty and some other conditions are fulfilled. Such a condition could be based on the global approximation error or on the number of vertices or tetrahedra already deleted.

3.3. Controlling the approximation error

As described in the last section we have to compute the error added to the mesh by a half edge collapse. Our mesh reduction algorithm has to be suitable for simplifying very complex meshes with some million tetrahedra. Therefore it is necessary to have an algorithm with linear memory and time consumption.

There are different types of errors which arise when collapsing an edge. The error types we consider in our cost function are the geometric error ΔE_g, the field error ΔE_f and the shape error of the tetrahedra ΔE_s. The total cost of an edge collapse is then computed as a weighted sum of the three components:

$$\Delta E(hec(\mathbf{p}_i \to \mathbf{p}_j)) = w_g * \Delta E_g + w_f * \Delta E_f + w_s * \Delta E_s. \quad (2)$$

The choice of the weights in our implementation is up to the user for individually tuning the results of the reduction process for his needs.

Geometric Error: We measure and control the geometric error with the help of error quadrics which were proposed by Garland and Heckbert [6]. The idea of the error quadrics is quite simple. The sum of the squared distance from a vertex $\mathbf{p} = [p_x p_y p_z 1]^T$ to the planes that meet at that point is

$$d(\mathbf{p}) = \sum_{\mathbf{e} \in planes(\mathbf{p})} (\mathbf{e}^T\mathbf{p})^2 \quad (3)$$

which can be rewritten as

$$\begin{aligned} d(\mathbf{p}) &= \sum_{\mathbf{e} \in planes(\mathbf{p})} (\mathbf{p}^T\mathbf{e})(\mathbf{e}^T\mathbf{p}) \\ &= \mathbf{p}^T\Big(\sum_{\mathbf{e} \in planes(\mathbf{p})} \mathbf{K_e}\Big)\mathbf{p} \\ &= \mathbf{p}^T\mathbf{Q}\mathbf{p} \end{aligned}$$

where $\mathbf{e} = [a,b,c,d]^T$ defines a plane $ax+by+cz+d=0$ where $a^2+b^2+c^2=1$. So the error quadric $\mathbf{Q}$ is represented as a 4×4 symmetric matrix which requires 10 float values to store.

To evaluate the costs ΔE_g of an half edge collapse on the boundary of the mesh we have to compute

$$\Delta E_g(hec(\mathbf{p}_i \to \mathbf{p}_j)) = \mathbf{p}_j^T(\mathbf{Q}_i + \mathbf{Q}_j)\mathbf{p}_j. \quad (4)$$

As mentioned earlier, because of using the half edge collapse we do not need to find an optimal position for the common vertex but just to classify the error introduced by the collapse.

Field error: Even more important than the geometric error is the field error of the scalar field defined by the tetrahedral grid. If the field error is not treated properly then this error will become apparent when visualizing the dataset on a coarse level.

When removing a vertex from the mesh by a half edge collapse the field is only changed in the 1-ring of tetrahedra around the removed vertex on level l. To find an appropriate error measure for the field error therefore only this region Ω has to be taken into account. As shown in [1] we can use the integral over the squared differences between the scalar values in the actual and in the original mesh over that region:

$$\Delta E_f(hec(\mathbf{p}_i \to \mathbf{p}_j)) = \frac{1}{|\Omega|}\int_\Omega |S(p) - S^{l-1}(p)|^2 dp \quad (5)$$

Evaluating this formula at the vertices of the original mesh which lie in Ω leads to a good approximation of the field error. But one problem is that the evaluation of the field error becomes slower while the grid reduction proceeds. Therefore we decided to use a local error estimation which only compares the region Ω before and after the half edge collapse. Again we approximate the evaluation by only evaluating the formula on the vertices of the finer level, in our case $\mathbf{p}_i$. So the simplified formula now is

$$\Delta E_f(hec(\mathbf{p}_i \to \mathbf{p}_j)) = \frac{1}{|\Omega|}|S^l(\mathbf{p}_i) - S^{l-1}(\mathbf{p}_i)|^2 \quad (6)$$

Shape error: For some applications not only the geometric and field error of the reduced mesh are essential. Especially when using simplified meshes for multilevel methods for numerical simulations the shape of the tetrahedra is very important.

There are several possibilities to compute a shape measure of a single tetrahedron, like the aspect ratio. Such a shape measure has to be invariant under translations, rotations and scaling. We need a measure with a predefined positive value for an equilateral tetrahedron and larger values for degenerated tetrahedra. As shown in [5] the condition number of the matrix **S** transforming an equilateral tetrahedron to an arbitrary tetrahedron T can be used for defining such a shape measure.

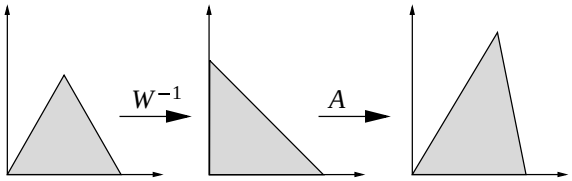

Figure 4. Transforming an equilateral to an arbitrary tetrahedron

As shown in Figure 4 the matrix $\mathbf{S} = \mathbf{A}\mathbf{W}^{-1}$ can be split into two matrices **A** and **W**. **A** transforms a right-angled reference tetrahedron into the tetrahedron T, **W** transforms the reference tetrahedron into an equilateral tetrahedron. W is constant and can be computed as

$$W = \begin{pmatrix} 1 & \frac{1}{2} & \frac{1}{2} \\ 0 & \frac{\sqrt{3}}{2} & \frac{\sqrt{3}}{6} \\ 0 & 0 & \frac{\sqrt{2}}{\sqrt{3}} \end{pmatrix}.$$

To get a shape measure for the tetrahedron element T we now have to compute the condition number $\kappa(\mathbf{S}) = \|\mathbf{S}\|\|\mathbf{S}^{-1}\|$ with the Frobenius norm of **S** defined by

$$\|\mathbf{S}\| = [tr(\mathbf{S}^T\mathbf{S})]^{\frac{1}{2}}$$

A tetrahedral shape measure is now defined by $M(T) = \kappa(S) - 3$. So an equilateral tetrahedron has a value of 1 and flat tetrahedra have a values of infinity in the limit case. To measure the shape error ΔE_s of the half edge collapse $hec(\mathbf{p}_i \rightarrow \mathbf{p}_j)$ we compute the average of the shape measures of all n triangles around the vertex $\mathbf{p}_j$, i. e.,

$$\Delta E_s = \frac{1}{n}\sum_{k=1}^{n} M(T_k)$$

4. Progressive Isosurfaces

Numerical simulations often cannot be computed on the local PC because of the speed and the memory needed for the computation. Therefore the simulations have to be computed on large computers which may only be connected by slow data connections to the local visualization computer. By transforming a tetrahedral grid into a progressive representation as shown in the last section, it is possible to start rendering the grid after the base mesh is completely transmitted. This base mesh is normally very small, only about 1% of the original grid. On this coarse grid it is also possible to start visualizing the data connected with the grid. In this section we will show how isosurfaces can be extracted progressively from the tetrahedron mesh.

4.1. Extracting isosurfaces from tetrahedral meshes

The extraction of isosurfaces from tetrahedral scalar fields is easier than from regular grids. For regular grids the standard approach for computing isosurfaces is the marching cubes algorithm [14]. Because of the eight corner vertices of an hexahedron we get 256 possibilities for the vertices to have greater or lesser values than the isovalue. Disregarding the symmetric cases, we end up with 16 different possibilities how the isosurface can intersect a hexahedron.

In the case of tetrahedra there only 16 different settings for the corner vertices and we end up with three different cases. In the first case, a tetrahedron is not intersected by the isosurface and nothing has to be done. In the next case three edges of the tetrahedron are cut by the isosurface which results in one triangle. In the last case four edges of the tetrahedron are intersected by the isosurface which will produce two triangles for the isosurface. The different cases can be seen in Fig. 5.

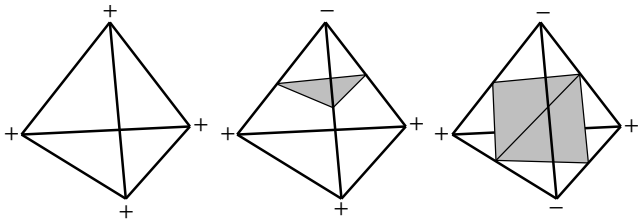

Figure 5. The three different cases for a tetrahedron to be intersected by the isosurface and the resulting triangles.

To compute the position of the vertices of the isosurface we linearly interpolate the scalar value between the vertices

$v_0 = \binom{\mathbf{p}_0}{s_0}$ and $v_1 = \binom{\mathbf{p}_1}{s_1}$ of the edge, i. e.,

$$\mathbf{q} = \frac{s_1 - s}{s_1 - s_0}\mathbf{p}_0 + \frac{s - s_0}{s_1 - s_0}\mathbf{p}_1 \tag{7}$$

In the third case, where we find four intersections of a tetrahedron, the triangulation is not unique.

4.2. Speeding up the isosurface extraction

If the base mesh is already very large, the initial isosurface extraction becomes very slow. Therefore we have to implement a technique so that it is not necessary to check all tetrahedra whether they are intersected by the isosurface. Quite a lot of work has already been done in this research area [16, 15, 2].

In a naive implementation each tetrahedron has to be checked if it is intersected by the isosurface. This happens if the smallest value s_{min} of a cell is lower than the isovalue s and the largest value s_{max} is higher. In [13] it is shown that all elements with a minimum value s_{min} and a maximum value s_{max} can be mapped in the so-called *span space*, an $\mathbb{R}^2$ value space. Each tetrahedron cell is assigned to a point position (s_{min}, s_{max}) in the span space. All tetrahedra which are intersected by the isosurface can now be found in a rectangular area above the $X = Y$ line, which is shown in Figure 6.

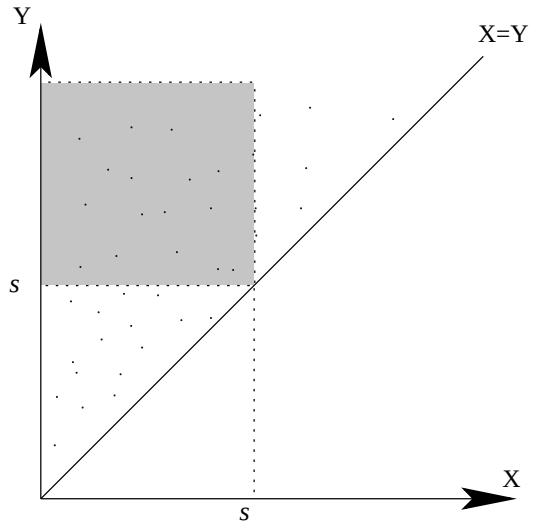

Figure 6. Elements in the Span Space.

What is needed now is an efficient strategy to search for all elements situated in the area shown. Therefore we divide the span space into subdomains and arrange them in an *interval tree*, as proposed from Cignoni *et al.* [2]. An interval tree is a binary search tree over interval (s_{min}, s_{max}) values. The interior nodes are assigned to a split value s. The split values in the left subtree are entirely less than s, in the right subtree entirely greater than s. There are two lists assigned to every node where tetrahedra are inserted if $s_{min} < s < s_{max}$. In the first list the tetrahedra are sorted in ascending order for the minimal value s_{min}. In the second list the same elements are sorted in descending order for the value s_{max}.

In [2] it was shown that the query complexity of the interval tree data structure is $O(k + \log(h))$, where k is the size of the output and h is the number of unique intervals in the tree. This means that the query complexity in this data structure does not depend on the number n of tetrahedra in the mesh.

4.3. Progressive Refinement of the Isosurface

Many multiresolution approaches for extracting isosurfaces work on the basis of different levels of detail. A fast isosurface extraction can be performed on a coarse level of detail. If the user has found the appropriate isovalue, a finer mesh is used to extract the final isosurface. In our approach we use the fact that by the transformation of the mesh into a progressive representation, we have given continuous levels of detail. Each of the vertex splits has only a limited support where changes in the mesh have to be computed. This means that also the isosurface will change in that small area of the mesh. So only a few more operations are necessary to simultaneously improve the isosurface while refining the mesh.

When a vertex is inserted by a vertex split there are only changes in the mesh in the 1-ring neighborhood around the vertices of the new edge. We only look at the case of half edge collapses. Therefore the start vertex $\mathbf{p}_0$ does not change its position and we have three different types of tetrahedra T in the neighborhood:

$$\begin{aligned}
\mathcal{T}_{old} &= \{T \in \mathcal{M} | \mathbf{p}_0 \in T \wedge \mathbf{p}_1 \notin T\} \\
\mathcal{T}_{new} &= \{T \in \mathcal{M} | \mathbf{p}_0 \in T \wedge \mathbf{p}_1 \in T\} \\
\mathcal{T}_{moved} &= \{T \in \mathcal{M} | \mathbf{p}_0 \notin T \wedge \mathbf{p}_1 \in T\}
\end{aligned}$$

Tetrahedra $\mathcal{T}_{old}$ around the start vertex do not change and the isosurface does not need to be updated. The new tetrahedra $\mathcal{T}_{new}$ around the edge have to be tested whether they are intersected by the isosurface. The third group are tetrahedra $\mathcal{T}_{moved}$ which were connected with the start vertex $\mathbf{p}_0$ before the split and are now connected with the end vertex $\mathbf{p}_1$ of the mesh. In these tetrahedra the isosurface may have changed and therefore has to be updated. These three regions are shown in Figure 7 for the 2D case.

Due to the linear interpolation of the vertices of the isosurface, only vertices on the edges which have $\mathbf{p}_1$ as one end vertex have to be computed. Vertices on other edges in

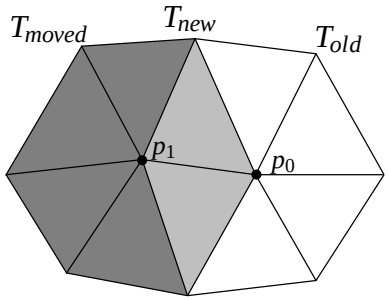

Figure 7. The three different regions around an edge inserted by a vertex split.

the configuration shown do not change their position. After computing the new vertex positions all tetrahedra around $\mathbf{p}_1$ are tested for intersection with the isosurface and the isosurface is updated locally. This means we recompute the triangles of the isosurface for those tetrahedra and delete or insert new triangles of the isosurface where necessary.

If the user changes the isovalue the complete isosurface has to be extracted again. To be able to use the acceleration described in the last subsection we have to update the interval tree when inserting new tetrahedra into the mesh. Here we use the same regions as for the progressive extraction of the isosurface. Tetrahedra in T_{new} are not in the interval tree and have to be inserted. For the tetrahedra in T_{moved} the max/min interval may have changed. In this case the interval tree has to be updated and the tetrahedra have to be moved to the correct node in the tree. An update for a tetrahedron will only occur when the max/min interval shrinks and s_{min} becomes larger than s or s_{max} smaller than s. For the update the tetrahedron is removed from the interval tree and inserted again with the new interval bounds.

5. Results

We have implemented the algorithms described in this paper with the library *gridlib*[12]. This is a C++ framework library for the management of 3D adaptive hybrid unstructured grids for integrated simulation and visualization. The gridlib is developed at the Computer Graphics Lab, University of Erlangen. The library runs on PC, SGI workstations and Hitachi supercomputers. The following experiments have been executed on a PC with an Intel PIII 700MHz processor and 512Mb memory.

We present results for several datasets: **Elec** which is a simulation of an electrical field with 26288 vertices and 142131 tetrahedra; **Harmonic** representing an analytical dataset for one of the spherical harmonic functions with an exponentially decreasing radial dependency with 262144 vertices and 1572864 tetrahedra; **Shuttle** which is a result of a airflow simulation over a space shuttle (190584 vertices, 1058775 tetrahedra).

First we applied the grid reduction algorithm to the electrical field dataset, which was reduced to 1288 vertices and 7737 tetrahedra. In Figure 8 we show a visualization of this dataset with an isosurface and a color-coded slice using a post-interpolatory transfer function. The quality of the reduction can be seen by comparing the coarse level with the full resolution of the dataset.

In Figure 9 we illustrate the progressive extraction of isosurfaces using the harmonic dataset. The isosurface was extracted on the coarsest level, corresponding to 5% of the vertices of the initial dataset. By inserting new tetrahedra to the mesh the isosurface is updated locally and improves fast. In the dataset with 30% of the original vertex there is nearly no visible difference compared with the original dataset. When changing the isovalue the surface has to be recomputed on the actual level. In Table 1 we compare the times for the recomputation of the isosurface for a specified isovalue by using the interval tree for optimization and without using the interval tree.

dataset size	triangles	time (int. tree)	time
5%	14718	0.238	0.376
10%	21538	0.341	0.659
15%	24590	0.418	0.993
30%	28482	0.466	1.579

Table 1. Isosurface extraction times on different levels of detail (in sec.) for the harmonic dataset.

In a further experiment we show the quality of the reduction algorithm for preserving the outer shape of the tetrahedral mesh. In Figure 10 we show the boundary triangle mesh of the **shuttle** dataset in coarse (1068 triangles) and high (45932 triangles) resolution. By using error quadrics for the approximation of the geometric error the results are comparable to [6]. By also considering the other error types for edge collapses on the boundary, the collapse with the smallest geometric error will not always be performed first. Therefore the quality of the approximation cannot be as good as in the original work. In Figure 11 we have extracted an isosurface from the shuttle dataset at a number of resolutions starting with about 5% of the original size and going up to about 50% of the number of tetrahedra. Here again the progressive improvement of the isosurface can be noticed.

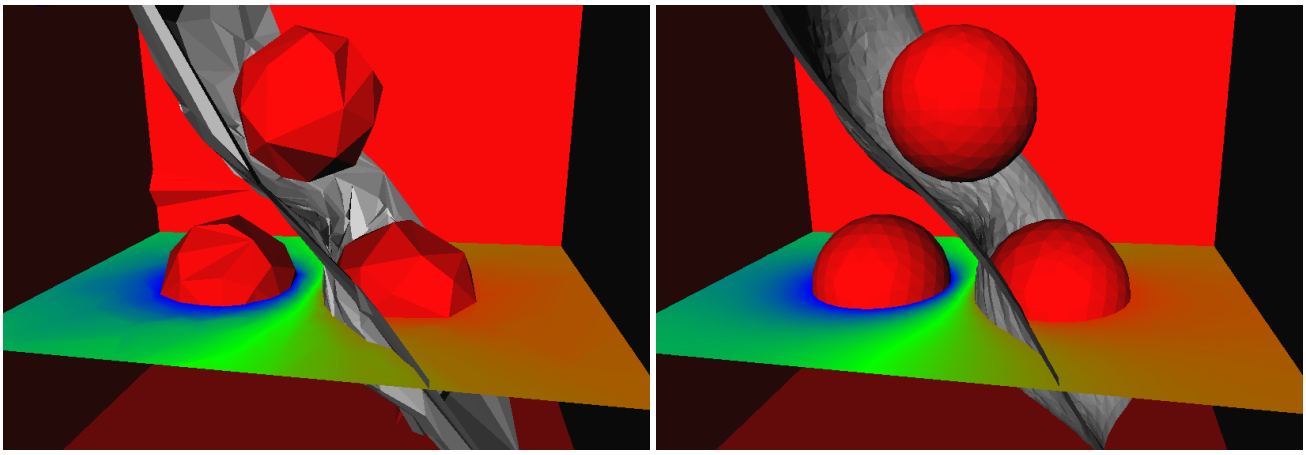

Figure 8. Visualization of the Elec dataset with an isosurface and a color coded slice, on the left in the coarsest and on the right in the finest resolution.

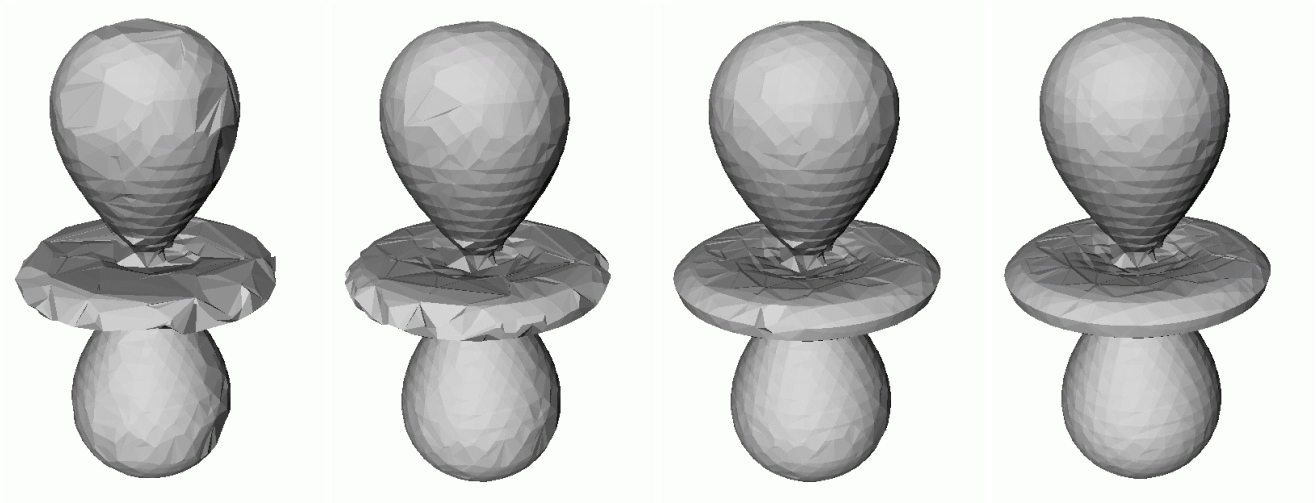

Figure 9. Progressive isosurface extraction in the harmonic dataset, from left to right with 5%, 10%, 16% and 30% of the original size of the dataset.

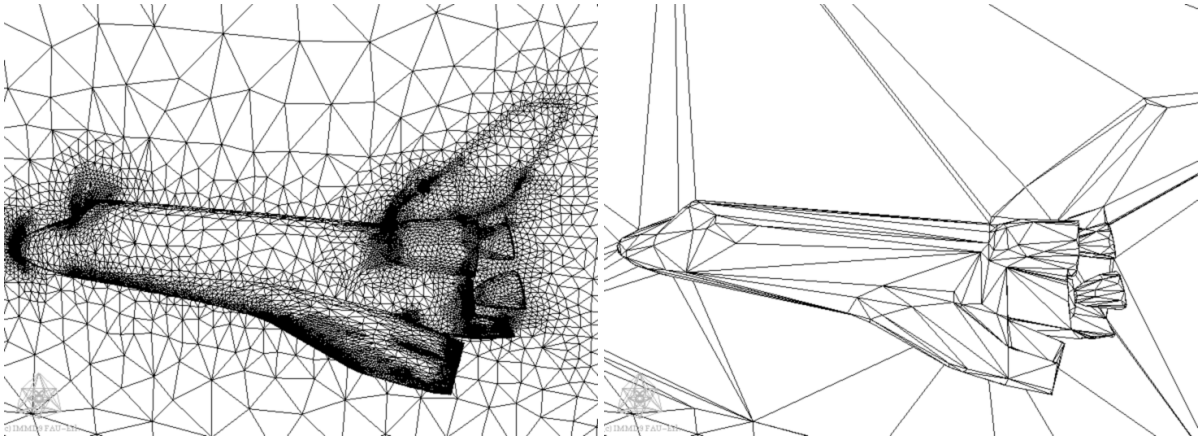

Figure 10. Shuttle: The outer shape of the space shuttle is not much influenced by the reduction algorithm.

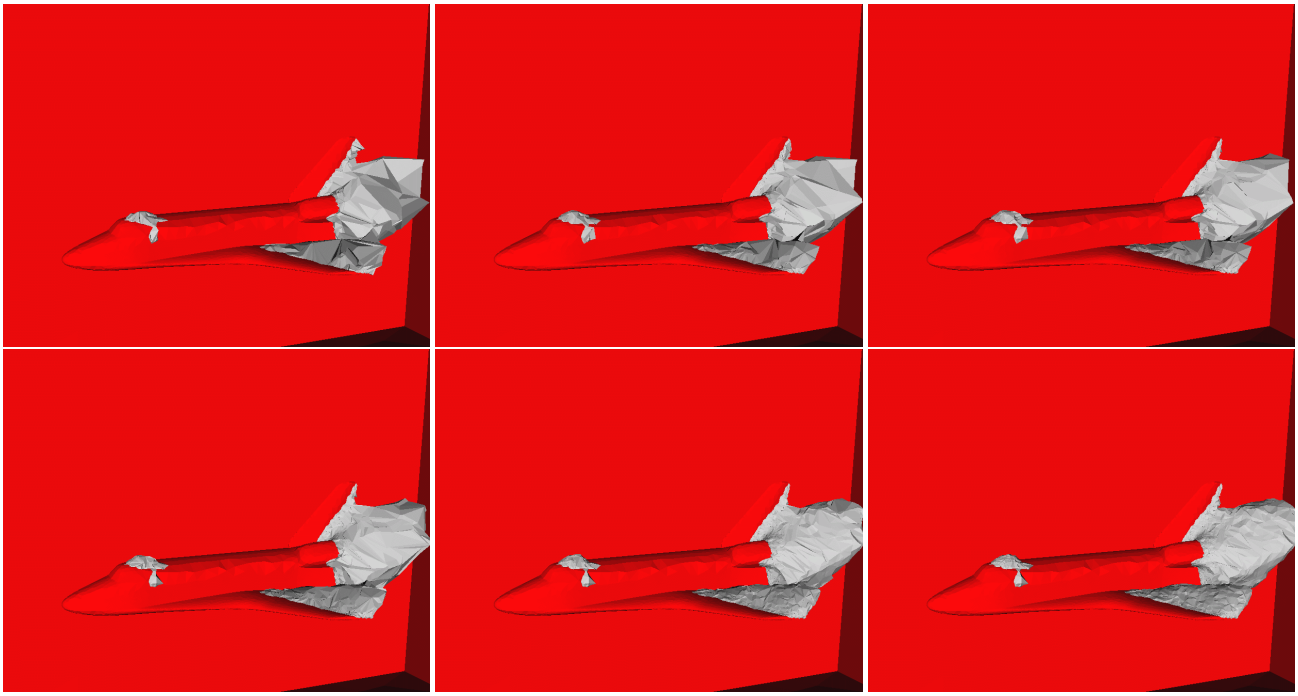

Figure 11. Progressive extraction of an isosurface for the space shuttle.

6. Conclusion

We have presented an efficient algorithm for extracting isosurfaces from a tetrahedral grid which is transmitted in a progressive representation based on half edge collapses. With the help of our approach it is possible to start visualizing a scalar field meshed with tetrahedra already after a small base mesh is transmitted. The quality of the visualization can be improved in continuous levels of detail by adding new vertices to the tetrahedral grid and locally adapting the visualization. In this paper we concentrated on the extraction of isosurfaces, but also many other visualization approaches like direct volume rendering can benefit from the progressive transmission. The ability to adapt the resolution of the mesh also makes it possible to achieve interactive frame rates on computers not equipped with high end graphic hardware.

Future work will concentrate on improving the quality of the coarse approximations of the tetrahedral grid. This includes developing more accurate error measures for scalar fields and also for vector fields which were not addressed in this paper.

Another interesting task is the parallelization of the algorithm. The grid reduction algorithm is time consuming because of the high computational costs of the error functions. We would like to use the parallel remote computer not only for the simulation but also for the generation of progressive mesh. First tests on a SGI Onyx with four processors have shown that setting up the priority queue has a speed up of factor three compared to using only one processor.

7. Acknowledgement

We would like to thank Manfred Kaltenbacher (LSE) for providing the electrical field dataset, Gunther Brenner (LSTM) for providing the space shuttle dataset and Michael Schrumpf for implementing the isosurface acceleration algorithms.

References

[1] P. Cignoni, C. Costanza, D.and Montani, C. Rocchini, and R. Scopignio. Simplification of tetrahedral meshes with accurate error evaluation. In *Visualization '99 Proceedings*. IEEE, 1999.

[2] P. Cignoni, P. Marino, C. Montani, E. Puppo, and R. Scopigno. Speeding up isosurface extraction using interval trees. *IEEE Trabsactions on Visualization and Computer Graphics*, 3(2):158–170, 1997.

[3] A. Doi and A. Koide. An efficient method for triangulating equi-valued surfaces by using tetrahedral cells. *IEICE Trans. Commun. Elec. Inf. System.*, E-74(1):214–224, 1991.

[4] K. Engel, R. Westermann, and T. Ertl. Isosurface extraction techniques for web-based volume visualization. In *Visualization '99 Proceedings*, pages 139–146. IEEE, 1999.

[5] L. A. Freitag and P. M. Knupp. Tetrahedral element shape optimization via the Jacobian determinante and condition number. In *Proceedings of the 8th International Meshing Roundtable*, 1999.

[6] M. Garland and P. Heckbert. Surface simplification using quadric error metrics. In *ACM Comp. Graph. (SIGGRAPH '97 Proc.)*, pages 209–216, 1997.

[7] M. Garland and P. Heckbert. Simplifying surfaces with color and texture using quadric error metrics. In *Visualization '98 Proceedings*, pages 263–269, 1998.

[8] R. Grosso and T. Ertl. Progressive iso-surface extraction from hierarchical 3d meshes. In *Comp. Graph. Forum (EUROGRAPHICS '98 Proc.)*, pages 125–135, 1998.

[9] A. Guéziec and R. Hummel. Exploiting Triangulated Surface Extraction Using Tetrahedral Decomposition. *IEEE Transactions on Visualization and Computer Graphics*, 1(4):328–342, Dec. 1995.

[10] H. Hoppe. Progressive Meshes. In *ACM Comp. Graph. (SIGGRAPH '96 Proc.)*, pages 99–108, 1996.

[11] H. Hoppe. New quadric metric for simplifying meshes with appearance attributes. In *Visualization '99 Proceedings*, pages 59–66, 1999.

[12] P. Kipfer. gridlib: System design. Technical Report 4, Universität Erlangen-Nürnberg, 2000.

[13] Y. Livnat, H. Shen, and C. Johnson. A near optimal isosurface extraction algorithm using the span space. *IEEE Transactions on Visualization and Computer Graphics*, 2(1), 1996.

[14] W. E. Lorensen and H. E. Cline. Marching cubes: a high resolution 3D surface construction algorithm. In M. C. Stone, editor, *SIGGRAPH '87 Conference Proceedings (Anaheim, CA, July 27–31, 1987)*, pages 163–170. Computer Graphics, Volume 21, Number 4, July 1987.

[15] H. Shen, C. Hansen, Y. Livnat, and C. Johnson. Isosurfacing in span space with utmost efficiency (issue). In *Visualization '96 Proceedings*, pages 287–294. IEEE, 1996.

[16] H. Shen and C. Johnson. Sweeping simplices: A fast isosurface extraction algorithm for unstructured grids. In *Visualization '95 Proceedings*, pages 143–150. IEEE, 1995.

[17] O. Staadt and M. Gross. Progressive tetrahedralizations. In *Visualization '98 Proceedings*, pages 397–402. IEEE, 1998.

[18] R. Westermann, L. Kobbelt, and T. Ertl. Real-time exploration of regular volume data by adaptive reconstruction of isosurfaces. *The Visual Computer*, (2):100–111, 1999.